中編

組み込みシステム設計者のためのLIN 2.0マイコン実装術

――使用するCPU性能に応じたオプション選び

舘 伸幸

前回（本誌2005年11月号，pp.92-97）に引き続き，プロトコル仕様について解説します．今回は，フレーム構造の詳細やスケジューリング，エラー処理を説明します（LIN Protocol Specification の2.1.2章以降の後半部分に相当）．また，診断に関する規定が記述されている仕様書である“Diagnostic and Configuration Specification”の内容も紹介します．（筆者）

フレームの先頭にはブレーク（break）と呼ばれるロー・パルス（“L”レベル）が存在します（**図1**）．LIN 1.3では同期ブレーク（sync break）という名称でしたが，LIN 2.0では単にブレークとなっています．

1．プロトコル詳細――フレームの設定方法

ブレークはロー・パルスが少なくとも13ビット以上であること，またスレーブは11ビット時間注1をしきい値として検出することが規定されています．ここでの13ビットは，マスタが±0.5％の精度で保証しなければなりません（この精度値は，“LIN Physical Layer Specification”で規定されている）．

一方，スレーブの検出しきい値は，スレーブ・ノードの動作クロックの周波数偏差を最大14％まで許容することから，11ビットで判別するようになっています．なお，プロトコル仕様書には，水晶などの精度の良いクロックを利用できるときは9.5ビットでの判定でもよい，という注意書きがあります．これは，フレーミング・エラーを利用してブレークを検出できる可能性を示唆しています．UARTの仕様にもよりますが，フレーミング・エラー発生時にUARTの受信バッファがゼロであれば，少なくとも9.5ビット以上のロー・パルスを受信したと言えるためです．

● フレーミング・エラーを使ってブレーク検出を簡便に

図2のUARTフォーマットを見てください．フレーミング・エラーとは，ストップ・ビットにおけるサンプリングで，“L”レベルが検出された場合に発生するエラーです（ス

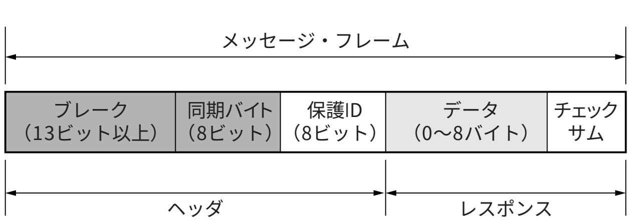


図1　フレームの構成

LINのメッセージ・フレームは大きくヘッダとレスポンスからなる．ヘッダはフレームの最初の部分（ブレーク，同期バイト，保護ID），レスポンスはデータとチェックサムで構成される．

注1：1ビット時間とは，あるネットワーク伝送速度における1ビットの伝送時間のこと．例えば，ネットワークの伝送速度が10kbpsの場合，1ビット時間＝$1 \div (10 \times 10^3) = 100\mu s$となる．

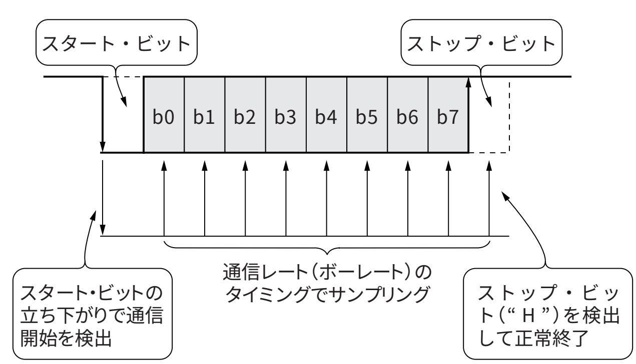


図2　UARTフォーマット

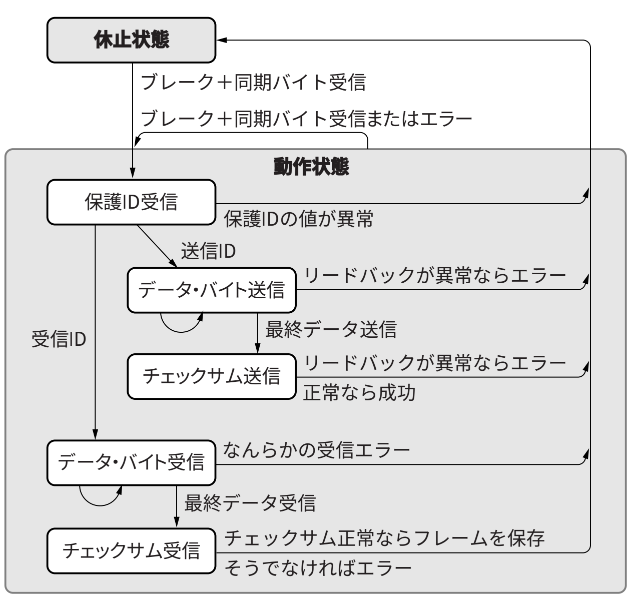
UARTのフレームでは，ストップ・ビットは1ビットの“H”レベルである．このストップ・ビットの位置で“L”レベルが検出されることをフレーミング・エラーという．

トップ・ビットは1ビットの“H”レベル）．フレーミング・エラーが発生するまでにサンプリングされた情報は，UARTの受信バッファ（シフト・レジスタ）に入っているので，受信バッファがゼロでフレーミング・エラーが発生したということは，すなわち，少なくとも9.5ビット（＝スタート・ビット1ビット＋データ・ビット8ビット＋ストップ・ビットのサンプリング位置までの0.5ビット）の幅のロー・パルスを検出できたと解釈できるわけです．

LIN 2.0では，ブレークと同期バイト（synch）が処理された後は動作状態（active）となります（**図3**）．そこからは，どこでもブレークを検出できなくてはなりません．ブレーク検出にフレーミング・エラーを利用できれば，簡便にこの仕様を実現できます．*RC*発振など，精度の良くない発振器の場合，休止状態（dominant）からのブレーク検出にフレーミング・エラーは使えませんが，フレーム途中のブレーク検出であれば（ボーレート補正直後なので）利用できる可能性もあります．

ただ，注記として「とくに必須ではない」との断り付きで，ブレークが部分的にデータ・バイトと重なった場合も検出できることが希望されるとあります．

この場合，フレーミング・エラーによる検出では対応できません．かといって，フレーム全域にわたってエッジ割り込みなどでロー・パルスのビット幅を検出し続けるという処理は，少なくともLINのスレーブ・ノードに使うような非力なCPUにとっては現実的ではありません．この「希望される仕様」を実現するには，ハードウェアに特別なロー・パルス検出回路を持たせるべきでしょう．低コスト・低負荷を考慮するならば，クラスタを設計する際にあえてこの仕様に挑戦しないという判断もあるでしょう．



※リードバック：送信したデータを自分自身で受信して整合をチェックすること

図3　状態遷移

ブレークと同期バイトが処理された後，動作状態となる．その後は，どこでもブレークを検出できなくてはならない（そのとき受信中のフレームはエラーとして廃棄する）．

● パルス幅の測定は時間的制約やCPU性能を考えて選択

ブレークの次に来る同期バイト（synch byte）は，通信レート同期のためのフィールドです．LIN 1.3では同期フィールド（synch field）という名称でした．

マスタ・タスクは，単にUARTで0x55を送信するだけです．一方，スレーブ・タスクは複数の処理手段が考えられます．水晶などの高精度の発振を利用していて十分な精度が見込まれる場合，単にUART受信を行って結果が0x55であることを確認するというのが，処理的にはもっとも簡単な方法です．精度の高くない発振回路を使用して仕様に忠実に設計する場合，タイマなどを利用してパルス幅を計測する処理が必要となります．

このあたりの詳細は“LIN Physical Layer Specification”の“Bit TIMING REQUREMENTS AND SYNCRONIZATION PROCEDURE”という章で記述されています．（理由はわからないが）パルス幅は立ち下がりエッジ間で計測しなければならないと記述されているので，実際の設計・実装でも仕様と一致しておいたほうがよいでしょう．また，

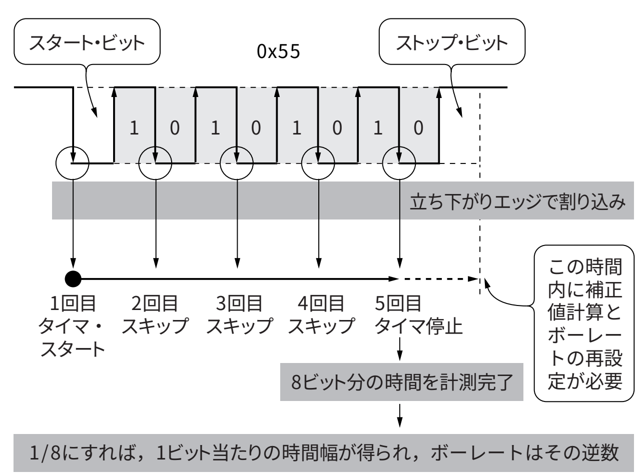


図4　ボーレート計測

仕様書ではスタート・ビットの立ち下がりからビット7の立ち下がりまでの8ビット分を計測して，1/8にする方法が推奨されている．

計算を簡単にするため，2，4，6，8ビット・タイムの距離で計測するとも記述されています．非力なマイコンでは比較的重い処理になるので，シフト演算1回で計算できる6ビット以外を選ぶとよいでしょう．

なお，プロトコル仕様書ではスタート・ビットの立ち下がりからビット7の立ち下がりまでの8ビット分を計測して，1/8にする方法が推奨されています(**図4**)．この場合，計測完了から同期バイトのストップ・ビットの終わりまで2ビット分の時間しかないので，とくに非力なマイコンの場合，ボーレート補正処理が同期バイトの次に来る保護IDの受信開始までに終了するかどうかを十分に検討しなくてはなりません．

十分な時間的余裕が見込まれるなら8ビットで，ややリスキーなら4ビットで計測というのが，実際の設計における選択肢になろうかと思います．

● 2種類のチェックサムで，V1.3とV2.0の混在が可能に

同期バイトの次のフィールドは保護IDです．保護IDは，IDとパリティで構成される1バイトのデータです．特別なIDとして，60(0x3c)と61(0x3d)がノード・コンフィグレーションとノード診断[注2]，およびスリープ要求に割り当てられています．0x3cをマスタからのリクエスト用，0x3dをスレーブのレスポンス用として使用します．

保護IDの後に1～8バイトのデータが来ます．LIN 2.0以前は，データ長はIDフィールドのビット4とビット5で指定していました(LIN 1.1では必須，LIN 1.3では推奨)．しかし，LIN 2.0ではクラスタの仕様で決定します．仕様書ではpublisher(レスポンスを送信するノード)とsubscriber(レスポンスを受信するノード)によって決定すると表現されていますが，これは送信する側と受信する側のアプリケーションの仕様で決定するという意味です．

フレーム最後のフィールドはチェックサムです．LIN 2.0では2種類のチェックサムを扱います．データ・バイトのみで計算するLIN 1.3と互換性のあるものを標準チェックサム(classic checksum)，保護IDバイトも計算に加えたものを拡張チェックサム(enhanced checksum)と言います．

スレーブ・ノードがLIN 1.3に対応している場合は標準チェックサムを，LIN 2.0に対応している場合は拡張チェックサムを使います．これはマスタ・ノードが管理します．仕様書群の中の“LIN Node Capability Language Specification”では，プロトコル・バージョンについての記述があります．これをもとに生成した情報にしたがって，マスタはノードごとのバージョンを把握します．つまり，LIN 2.0に対応したマスタを持つLINクラスタでは，LIN 2.0とLIN 1.3のスレーブ・ノードを混在して接続することができます．

一方で，後述するイベント・トリガ・フレームを使用したいクラスタでは，LIN 1.3とLIN 2.0を混在させてはならないという規定もあるので，注意が必要です．

●「空き時間」も考慮してスロット時間を定める

各フレームは，「スロット(slot)」という時間枠の中で送信されます．プロトコル仕様書では“2.2 Flame Slot”という章の中で記述されており，具体的な時間については以下の(1)～(6)の六つの式が定義されています[注3]．

$$T_{Header_Nominal} = 34 T_{Bit} \quad \cdots\cdots (1)$$

$$T_{Response_Nominal} = 10\,(N_{Data} + 1) \cdot T_{Bit} \quad \cdots\cdots (2)$$

$$T_{Frame_Nominal} = T_{Header_Nominal} + T_{Response_Nominal} \quad \cdots\cdots (3)$$

$$T_{Header_Maximum} = 1.4\, T_{Header_Nominal} \quad \cdots\cdots (4)$$

$$T_{Response_Maximum} = 1.4\, T_{Response_Nominal} \quad \cdots\cdots (5)$$

$$T_{Frame_Maximum} = T_{Header_Maximum} + T_{Response_Maximum} \quad \cdots\cdots (6)$$

公称値(nominal)は，フレームの内容を構成するビット数で定義されています．フレームには，**図5**に示すように，ヘッダからレスポンスまでの空き時間「レスポンス・スペース」や，フレーム終了から次のフレーム(ヘッダ)送信までの空き時間「インター・フレーム・スペース」などの“すき間”が含まれます(詳細は本稿の前編を参照)．さらにデータとデータの間にはバイト・スペースという空き時間も見込まれます．フレーム・スロットの公称値には，これらの空き時間は含みません．

そういう意味では，式(1)，(2)で示している値は，スロットの最小時間と言えるでしょう．

式(1)では，ヘッダの値を求めています．1ビット時間の34倍です．ヘッダ・フレームにはブレーク，同期バイト，保護IDが含まれますが，同期バイトと保護IDはそれぞれ10ビット(スタート・ビット1ビット＋データ8ビット＋ストップ・ビット1ビット)なので，それらを差し引くとブレークには14ビットが割り当てられます．これは，上

注2：詳細はLIN Diagnostic and Configuration Specificationで規定されている．
注3：ここでの式(1)～(6)は，プロトコル仕様書の“2.2 Flame Slot”の式(3)～(8)に対応していることに注意．

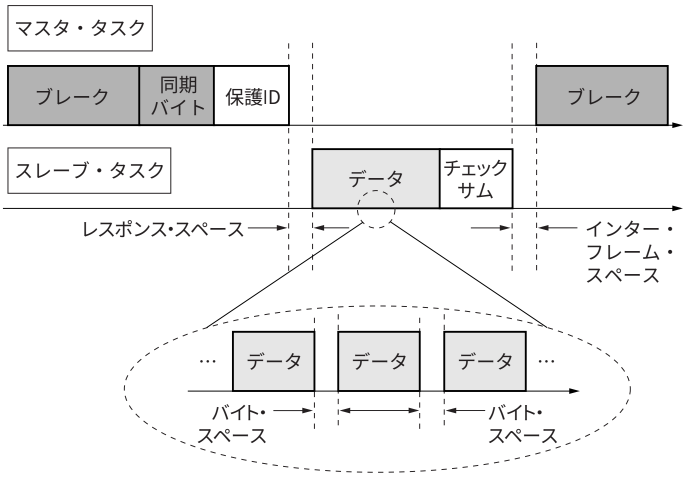


図5　フレームの空き時間
フレームには，レスポンス・スペース，インター・フレーム・スペース，バイト・スペースといった空き時間が存在する．

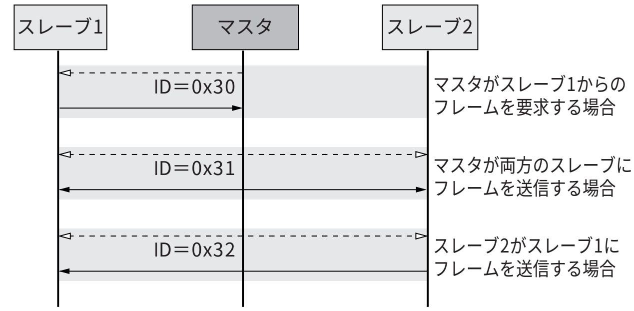


図6　無条件フレーム
通信はいつもマスタによって開始される．通信は一つのpublisher（レスポンスを送信するノード）と，それを受信する一つ以上のノードで成り立つ．

述した最低13ビット幅のロー・パルス（ブレーク）と1ビット分のブレーク・デリミタ（ブレークと次に続く同期バイトの境界のための“H”レベル1ビット）を意味します．

式（2）の最初の10は，同期バイトや保護IDと同じく，1バイト通信に要するビット数を表しています．N_{Data}（レスポンス中のデータのバイト数）＋1となっているのは，チェックサム分を加えているからです．式（1）と式（2）を足したものが式（3）であり，フレームを扱うのに最低限必要なスロット時間となります．

実際の通信では処理の遅れなどを考慮して，40％のマージンを加えます．それが式（4）～（6）になります．フレーム・スロットを設計する際は，この40％余裕を加えた式（6）の値かそれ以上の時間を確保する必要があります．

ブレークの説明ではロー・パルスは「少なくとも13ビット幅」としており，上限についての記述はありませんでした．しかし，この式（4）で示される値がヘッダの上限となるので，おのずとブレークの上限も制約を受けることになります．

また，仕様書には「すべてのノードはゼロ・オーバヘッドのフレームを受信できなくてはならない」という注意書きがあります．おそらく，この点についてもっとも留意が必要になるのはインター・フレーム・スペースです．確度の高い発振器を利用して，ブレーク検出にUARTのフレーミング・エラーを利用するような場合は，それほどシビアではないでしょう．しかし，別の回路でブレーク検出を行わせたり，UARTに（ブレーク受信のための）特別な指定や設定が必要な場合，フレーム受信完了後にチェックサムを計算したり，次のブレーク受信のための手続きにもたもたしていると，処理が間に合わなくなってしまいます．とくにスレーブに使うような非力なCPUの場合，フレーム間の処理タイミングについて十分に検討しておく必要があります．

● 5種類のフレームを用途に応じて使い分ける

LIN 2.0では，目的に応じて次に挙げる五つの種類のフレームを利用します．ただし，すべての種類をサポートする必要はないとしています．なお，予約フレーム（reserved frames）が定義されていますが，LIN 2.0では使用しません．

1）無条件フレーム（unconditional frame）

無条件フレームは，普通にノード間でデータをやりとりする場合に使います．

図6に，通信シーケンスの例を示します．図のいちばん上の例では，まずマスタがID＝0x30でヘッダを送信します．スレーブ1はこのIDにレスポンスを返し，スレーブ2はこのIDを無視しています（図ではID＝0x30のヘッダはスレーブ1のみに送信されているように見えるが，バス接続なので物理的にはスレーブ2にも届いている）．

2番目の例はマスタ・ノードが全スレーブにブロードキャストする場合です．マスタ・タスクのスレーブ送信に続き，マスタ・ノードのスレーブ・タスクがレスポンスを送信します．スレーブ1とスレーブ2は，このときのID（＝0x31）をレスポンス受信用として解釈できる（する）ように設定されている必要があります[注4]．

注4：LIN 1.3ではIDの値で送受信が決まっていたが，LIN 2.0はクラスタの仕様で決めることができる．

3番目の例はスレーブからスレーブへのデータ送信です．マスタがヘッダを送信すると，このときのID（＝0x32）でスレーブ1はレスポンス受信モードになり，スレーブ2はレスポンス送信モードになります．こうすることで，スレーブ・ノードどうしの通信が可能となります．

2）イベント・トリガ・フレーム（event triggered frame）

イベント・トリガ・フレームは，LIN 1.3ではコンフィグレーションの章に項目があっただけでしたが，LIN 2.0で動作が定義されました．LINは基本的にバスの衝突が発生しないしくみになっていますが，このフレームではバスの衝突が起こる可能性があり，それを許可しています．

このフレームは，複数のスレーブに割り当てられた同一機能が，同時に動作することがあまりない場合に使用します．仕様書では，このフレームを用いる例としてドアロック・ノブの監視が挙げられています．かりに無条件フレームを使うとすると，ドアの数だけ問い合わせを行う必要があり，例えば4ドアであればフレーム・スロットを四つ消費することになります．しかし，複数のドアロック・ノブが同時に操作されることはまれです（そもそも，一つのノブが操作されていることすらめったにない）．したがって，「変更があったら手を挙げて」と，すべてのドアに対していっせいに問い合わせれば，ほとんどの場合は1フレーム・スロットですみます．

動作の詳細を**図7**に示します．

最初のフレーム・スロットで，マスタはイベント・トリガ・フレームを送信しています．無条件フレームですべてのノードにブロードキャストする場合と似ていますが，ここではヘッダのみでレスポンスはマスタ・ノード（のスレーブ・タスク）からは送信しません．

図7の最初の例では，不運なことに（？）スレーブ1とスレーブ2の両方でイベント（例えばドアロック・ノブの操作）があった場合を示しており，両方からレスポンスが返されています．マスタはこのときバス上の衝突を検出して，複数のスレーブ・ノードでイベントが発生したことを知ります．この場合は，やむを得ず従来の方法で無条件フレームを使ってノード一つずつに問い合わせを行います（あるいは，この場合は次に述べるスポラディック・フレームを使うのが合理的）．これが，**図7**の2番目と3番目の通信例に相当します．

4番目の通信は，マスタからの問い合わせ時にどのノードでもイベントが発生していなかった場合です．このときはすべてのノードがレスポンスを返さず，マスタはフレーム・スロットのタイムアウトでイベント発生がなかったことを認識します．5番目は特定のノード（この場合はスレーブ2）でイベント発生があった場合です．スレーブ2のみがレスポンスを返しています．

3）スポラディック・フレーム（sporadic frame）

スポラディック・フレームは，マスタからの散発的な要求をスレーブに伝えるためのしくみです．

仕様書には，スポラディック・フレームはかならずデータを搬送すると記述されています[注5]．特定のIDに割り当てられた信号に変更があった場合，マスタはそれを検知した時点でもっとも近い空きスロットを使い，スポラディック・フレームでクラスタに送信します．では，スポラディック・フレーム用にあらかじめフレーム・スロットを用意しておく必要があるのでしょうか．これについては，後で詳しく述べます．

また，通常のケースとして複数のスポラディック・フレームが同じフレーム・スロットに割り当てられている場合の規定もあります．つまり，変更を通知するべき信号を含むスポラディック・フレームは複数あっても，スポラディック・フレーム用のスロットは1種類だけ用意する場合です（**図8**）．そもそも頻繁に発生することがわかっているなら，無条件フレームでスロットのプランを考えておくべきなので，これで十分なのでしょう．たまたま同時に複数の

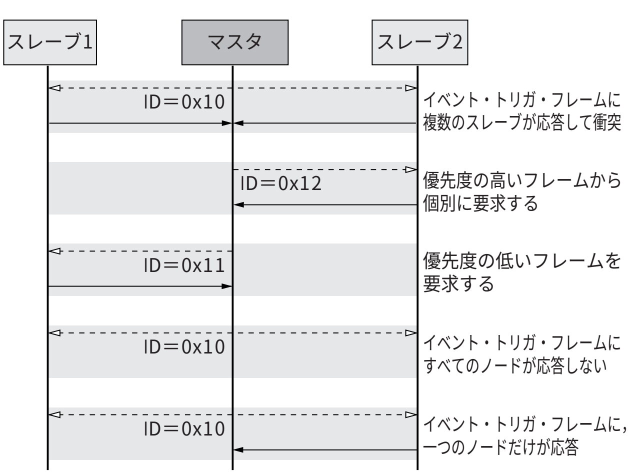


図7　イベント・トリガ・フレーム

0x10は，無条件フレームである0x11と0x12のフレームに関連するイベント・トリガ・フレーム．図中のフレーム・スロットは連続している必要はなく，途中にスケジュール定義された別のフレーム・スロットがあってもよい．

注5：例えば，イベント・トリガ・フレームでは，ヘッダのみでレスポンス（データ）がない場合もある．

スポラディック・フレームを使用する信号の変更があった場合，それらの信号の優先度に従って送信します．

また，スポラディック・フレーム用のスロットが巡ってきたときに，関連する信号の変更が何もない場合は，空きスロットのままにしてマスタは何も送信しません．**図9**の例は，マスタが信号の変更を通知する場合，つまりマスタ・ノード（のスレーブ・タスク）がレスポンスを送信する場合のみを示しています．しかし，無条件フレームと同じように，publisherがスレーブ・ノードの場合もありえます．

4）診断フレーム（diagnostic frame）

診断フレームは，ノード・コンフィグレーションと診断用にIDが割り当てられているフレームのことです．診断の要求には0x3cを使用し，診断結果の吸い上げには0x3dを使用します．詳細は仕様書群の中の“ Diagnostic and Configuration Specification ”に記述されています．

5）ユーザ定義フレーム（user-defined frame）

ユーザ定義フレームは，クラスタとして何か特別な機能を持ったフレームを追加定義したいときに使用します．IDは0x3e一つだけなので，追加定義できるフレームは1種類です．

2．プロトコル詳細――スケジュール，エラー処理

LINプロトコルの主要な特性はスケジュール・テーブルを使用することです．これにより，クラスタは時間軸で保証された動作を行えます．ただし，仕様書ではスケジュールを乱す要因が二つ規定されています．一つはイベント・トリガ・フレーム，もう一つはスポラディック・フレームです．

● スロットのスケジューリングはクラスタ仕様で決まる

イベント・トリガ・フレームにより複数のスレーブが応答してバス上で衝突が発生してしまった場合，マスタは無条件フレームで個々のスレーブに対して問い合わせを行って，これを解決しなければなりません．ということは，バス衝突に伴ってスレーブ・ノードの数だけ臨時にスロットが必要になるわけです．だからといって，それだけの空きスロットをあらかじめスケジュールしておいたのでは，少なくとも時間的メリットはなくなってしまうように思われます[注6]．

一方，スケジュールを変更してフレームを挿入すれば，クラスタの時間保証が壊れてしまいます．これはスポラディック・フレームも同じで，散発的な信号の変更などが起こるたびにスロットを挿入すれば，それだけ後のスケジュールに遅れが出てしまいます．

このことについてプロトコル仕様書の“ SCHEDULES ”では，「スポラディック・フレームやイベント・トリガに関連づけされる無条件フレームはスケジュール・テーブルに割り当てられない場合がある」と記述しています．

つまり，ここはクラスタの仕様に依存することになります．スポラディック・フレームの使用条件の成立や，イベ

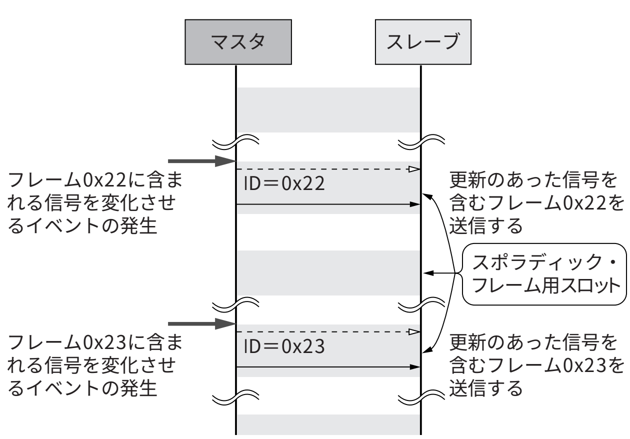


図8　複数のスポラディック・フレーム

まれに変更される信号を含むスポラディック・フレームが複数ある場合，それぞれに専用のフレーム・スロットを用意する必要はない．スポラディック・フレーム用の汎用スロットをスケジュールしておき，信号の変更イベントに応じていちばん近いスポラディック・フレーム・スロットを利用する．

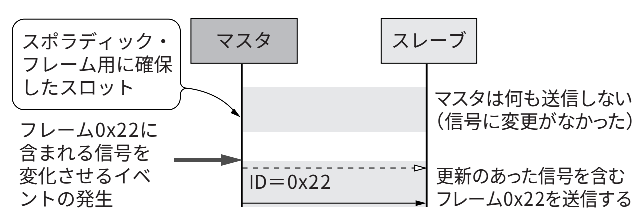


図9　スポラディック・フレーム

通常，スポラディック・フレームのためのスロットは空（何も通信されない）である．図の2番目のスロットでは，それ以前に0x22のフレームに含まれる信号の更新イベントがあったため，スポラディック・フレームが送信されている．図中のフレーム・スロットは連続している必要はなく，途中にスケジュール定義された別のフレーム・スロットがあってもよい．

注6：例えば，四つのスレーブ・ノードがクラスタに接続されているとする．イベント・トリガ・フレームで問い合わせを行う場合，衝突を検出すると無条件フレームを4回出す必要がある．だからといって，イベント・トリガの後にスケジュール・テーブルに四つの空きフレーム・スロットを作ったのでは，衝突が起こらなかった場合にその時間は何もできないため，むだになる．

ント・トリガによる衝突によって起こるスケジュールの変更（の頻度）が，クラスタの時間的仕様のマージン内かどうかで判断します．時間的にそれほど厳しくないクラスタであれば，臨時のフレームに対してスケジュール変更するとか，スロットを挿入するといった方法が簡単かもしれません．一方，時間周期を厳密にしたい場合，あらかじめ空きスロットをスケジュールしておくべきでしょう．

イベント・トリガで衝突が発生した場合，スポラディック・フレーム・スロットで順番にスレーブにアクセスしていくというのも一つの方法です．**図7**の例では，最初のスロットで衝突が発生し，2番目と3番目のスロットで各スレーブに個別にアクセスする場合を示しました．この2番目と3番目のスロットのような個別アクセス用に，スポラディック・フレーム・スロットを利用することができます．この場合，イベント・トリガによる衝突時のフォローのためだけに空きスロットをスケジュールする必要がなくなり，スロット・スケジュールをより効率的に運用することができます．

● エラー状態のクリアと復帰が重要なポイント

“TASK BEHAVIOR MODEL”という章では，マスタ・タスクとスレーブ・タスクのビヘイビア・モデルが示されています．

図3に示したスレーブの状態遷移図を再度ご覧ください．ここでのポイントは二つです．まず，エラー発生時はすべてブレーク待ちの休止状態に戻るという点です．設計時には初期化処理など，エラー状態を完全にクリアして確実に次のフレームに対応できるよう復帰する処理が重要でしょう．もう一つは，フレーム受信中はBreakAndSynchをいつでも受け付けることができるということを明示している点です．この動作の設計は，クラスタの仕様からよく考慮する必要があります．つまり，ブレーク受け付けタイミングをどうするかです（詳細は，上述のブレークに関する箇所を参照）．

● WakeUp信号検出はマイコンだけでは困難な場合もある

“NETWORK MANAGEMENT”という章には，スリープとウェイクアップについて記述されています．

1）ウェイクアップ（wake up）

ウェイクアップ要求は，250μs～5msの間，バスをドミナント状態（“L”レベル）にすることで行います．これは，あるボーレートにおける5ビット分のロー・パルスであり，UARTで0xF0を送信することに相当します（例えば，250μsは20kbpsにおける5ビット幅の“L”に相当する）．

起こされる側のスレーブ・ノードは，150μs以上のドミナント・パルス（ロー・パルス）をウェイクアップ信号と認識して，ドミナント・パルスの終了（すなわち立ち上がり）から100ms以内に通常の動作状態になる必要があります．

最初の150μsのドミナント・パルスの認識ですが，マイコン単体では実現困難な場合があります．なぜなら，マイコンがスリープから復帰する際に主発振器の発振安定時間などでms単位の時間が必要なことがあるためです．この場合，ウェイクアップ信号の処理は，物理回線のドライバ回路を用いて行うしかありません．立ち上がりからの100msはわざわざ計測する必要はなく，この時間内にマイコンが正常動作に戻ることが保証できていればよいだけです．

スレーブからウェイクアップ要求を出した場合，150ms以内にマスタからのヘッダがない場合はリトライします．3回リトライしてもだめな場合，4回目の送信まで少なくとも1.5秒待たねばなりません．

2）強制スリープ（go-to sleep）

ID＝0x3cで，データの最初のバイトが0x00のフレームが，クラスタに対するスリープ命令です．この命令が出された場合以外でも，バスが4秒以上動かなければ各ノードは勝手にスリープします．

● イベント・トリガ・フレームのエラー状態に注意

“STATUS MANAGEMENT”の章では，動作中のエラー検出のためのしくみについて規定しています．この規定は，次の二つを実現することを目的としています．

- 故障ユニットを容易に取り替えられるようにする
- 障害時にノードがlimp home modeに入るようにする

“limp home mode”は聞き慣れないことばですが，これは動作クロックを落としたり，重い処理を止めたり，通信に最低限必要な機能以外を停止したりすることでなんとか障害をマスタに伝えたりする，クラスタとしての非常時動作仕様です．

最初にコンセプトが記載されています．そこでは，マスタがクラスタのステータスを管理することと，各ノードがLINバス上の通信をモニタすること，それによって自身の動作不良を発見して対処する（limp home modeになる）としています．

次の注意事項では，イベント・トリガ・フレームにおける衝突時に，フレーミング・エラーが起こったからといって，それをエラーと判断してはならないとしています．なぜなら，LIN 2.0ではイベント・トリガ・フレームでバスの衝突を許容しているからです．

具体的なエラー・レポートについても規定されています．各スレーブは，Response_Errorというビットを送信フレーム（レスポンス）に入れることで，検出したエラーをマスタにレポートします．このResponse_Errorのビット位置は，ノード記述によって設定できるようになっています（詳細は仕様書群の中の“Node Capability Langu-age Specification”で規定）．

スレーブ・ノードは，受信時と送信時にエラーをチェックしてResponse_Errorをセットします．受信時のエラーはチェックサム・エラーやフレーミング・エラーであり，送信時のエラーはビット・エラー（自分が送信したものを同時に受信しても一致しない状態）が主です．

基本的に，イベント・トリガ・フレーム以外のすべてのフレームのどれか一つにResponse_Errorビットを設定することができます．イベント・トリガ・フレームに対してはResponse_Errorビットを設定してはなりません．その理由は，ヘッダに対してレスポンスが返ってこないというのは重要なエラー情報ですが，イベント・トリガ・フレームではレスポンスが返ってこなくても正常動作とみなせるためです．

最後にノードそのものの内部エラー管理について規定しています．error_in_responseとsuccessful_transferの2ビットで管理します．error_in_responseは，Response_Errorと同じ条件でセットされると考えてよいでしょう．successful_transferは，送受信が完了したらセットされます．つまり，error_in_responseが‘0’で，successful_transferが‘1’であれば，自ノードは正常に動作していると判断できるわけです．それ以外の組み合わせでは，なんらかの障害が検出されていることになります．ノードのアプリケーションはこの情報をもとにエラーを検出したら，なんらかのアクションを起こす必要があるでしょう．これはアプリケーションごとの，あるいはクラスタの仕様として規定すべき内容となります．

3．ノードの診断とコンフィグレーション

最後に，“LIN Diagnostic and Configuration Specification”について簡単に触れておきます．

● スレーブに機能を割り当てるコンフィグレーション

本仕様書には，ノード・コンフィグレーションに関する規定が記されています．ノード・コンフィグレーションは，マスタから保護IDを各スレーブ・ノードの持つ機能に割り振り，LINプロトコルにのっとった通信を行うためのしくみです．

本仕様書には，**図10**のようなスレーブ・ノードのメモリ・モデルが示されています．**図10**のNADは，各スレーブ固有のノード・アドレスを意味します．仕様書では，注記として初期NADはジャンパ線で設定する例が示されていますが，試作品などではDIPスイッチを用いる場合もあるでしょう．ROMにはLINプロダクトIDとして，メーカ（サプライヤ），機能，型番の情報を格納しておきます．メーカを示す値は固有の番号が割り振られており，これはLINコンソーシアムが管理しています．機能や型番はメーカが割り振ります．また，ROMにはそのスレーブ・ノー

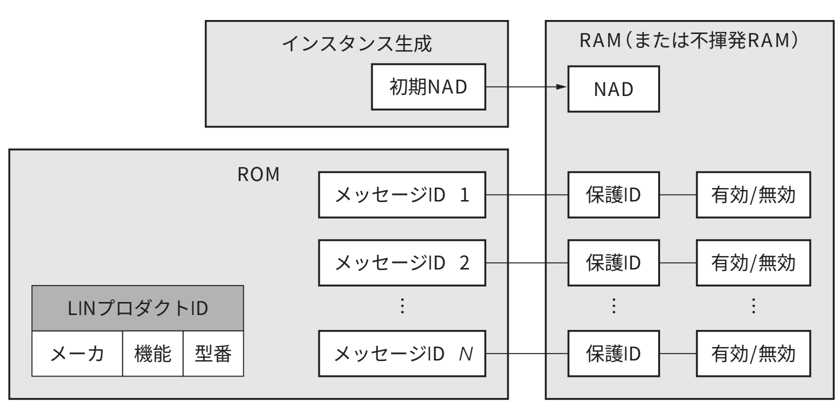


図10
スレーブ・ノードのメモリ・モデル
初期NADはジャンパやディップ・スイッチで設定され，ノード起動時にRAMに読み込まれる．

表1 保護IDの関連づけ

NAD	PCI	SID	D1	D2	D3	D4	D5
NAD	0x06	0xb1	メーカID 下位バイト	メーカID 上位バイト	メッセージID 下位バイト	メッセージID 上位バイト	保護ID

表2 プロパティ読み出し

NAD	PCI	SID	D1	D2	D3	D4	D5
NAD	0x06	0xb2	保護ID	メーカID 下位バイト	メーカID 上位バイト	機能ID 下位バイト	機能ID 上位バイト

表3 ノード・コンフィグレーション用のフレーム

<table>
<tr><th>仕様書の項目</th><th>記述内容</th></tr>
<tr><td>2.2 WILD CARDS</td><td>NADとプロダクトIDにはワイルド・カードが使える（NADにワイルド・カードを使用するのは，ブロードキャストを意味する）
<table>
<tr><th>プロパティ</th><th>ワイルド・カードの値</th></tr>
<tr><td>NAD</td><td>127（0x7F）</td></tr>
<tr><td>メーカID</td><td>0x7FFF</td></tr>
<tr><td>機能ID</td><td>0xFFFF</td></tr>
<tr><td>メッセージID</td><td>0xFFFF</td></tr>
</table></td></tr>
<tr><td>2.3 PDU STRUCTURE</td><td>診断フレームで送信されるフレームをPDU（Packet Data Unit）と呼ぶ．また，マスタが発行するフレームをリクエスト，スレーブが発行するフレームをレスポンスと呼び，かならず8バイトで送信される．
<table>
<tr><td>リクエスト</td><td>NAD</td><td>PCI</td><td>SID</td><td>D1</td><td>D2</td><td>D3</td><td>D4</td><td>D5</td></tr>
<tr><td>レスポンス</td><td>NAD</td><td>PCI</td><td>RSID</td><td>D1</td><td>D2</td><td>D3</td><td>D4</td><td>D5</td></tr>
</table>
●NADの値が0x00の場合はスリープ・コマンド，0x01～0x7Eの場合はスレーブ・ノード・アドレス，0x7Fの場合はブロードキャストを意味する．
●PCIは有効データ長を表す（上位4ビットはゼロ，下位4ビットが有効データ長）．データ長はSID/RSIDを含む長さのことで，例えばD1とD2が意味あるデータでD3～D5が無効なデータの場合，PCI＝0x03となる．
●SIDはISO 15765-3に準拠した値．ノード・コンフィグレーションには0xB0～0xB4が使用される．
●RSIDはレスポンス時のSIDを意味する．リクエスト・フレームのSIDに0x40を加えた値（0xF0～0xF4）．
●D1～D5はデータ・バイト．無効なバイトはすべて‘1’で埋める（0xFFとする）．</td></tr>
<tr><td>2.4　LIN PRODUCT IDENTIFICATION</td><td>プロダクトIDの構造は右のとおり．
<table>
<tr><th>D1</th><th>D2</th><th>D3</th><th>D4</th><th>D5</th></tr>
<tr><td>メーカID
下位バイト</td><td>メーカID
上位バイト</td><td>機能ID
下位バイト</td><td>機能ID
上位バイト</td><td>型番</td></tr>
</table></td></tr>
</table>

ドが扱うフレームごとのメッセージIDが格納されます．一方，RAMにはメッセージIDに対応した保護IDの変数があり，初期状態では無効（invalid）状態になっています．

スレーブ・ノードは起動時に初期NADをRAMにコピーし，自分のNADとします．この状態でマスタから保護IDが割り当てられるのを待ちます（マスタの保護ID割り当てについては，本仕様書の“2.5.1 Assign frame identifier”に記述されている）．保護ID割り当ての際にマスタから送信されるフレーム（リクエスト）の構造を**表1**に示します．このリクエストでNADとメーカIDが一致したスレーブ・ノードの指定メッセージIDが，D5で指定される保護IDに関連づけられます．このリクエストへのレスポンスはオプションです．スレーブ・ノードのプロパティを読んで，保護IDへの関連づけがうまくいったかどうかを確認することも可能です（**表2**）．また本仕様書では，ノード・コンフィグレーションに用いるフレームを定義しています（**表3**）．

● 三つの診断方法が規定されている

本仕様書の第3章では，ノードの診断情報を収集する方法が規定されています．三つの方法が記載されていますが，すべてオプションです（実装が必須ではない）．

まず一つ目は，“SIGNAL BASED DIAGNOSTICS”です．これは，通常の無条件フレームを使って診断情報の送信を行う方法です．診断のための特別な処理が不要であり，スレーブ・ノードにとってオーバヘッドが低いという利点があります．その一方で，扱うデータは（無条件フレームの）フレーム構造で制限を受けます．

二つ目はユーザ定義の診断フレームです．診断フレームの構造は，**表3**に示したとおりです．1バイト目のNADに未定義の128（0x80）以上の値を割り振ることによって，自

Column コンフォーマンス・テスト

「仕様どおり作ったはずなのに通信できない」，「ときどき動作がおかしい」といったことを防ぐため，LINプロトコルにはコンフォーマンス・テスト（conformance test）という第三者認証制度があります．

プロトコル仕様に対応したコンフォーマンス仕様に基づき，物理層やデータリンク層，ノード・コンフィグレーション，EMC（electro magnetic compatibility）の各項目について検査が行われます．これによって，各ECU（ノード）は一つの基準で判断された品質を保証できます．

LINのコンフォーマンス・テストでもっとも有名な認証機関は，ドイツFachhochschule Wolfenbüttel大学の研究組織C＆S（Communication & Systems Group）であり，国内ではアドバンスド・データ・コントロールズ（ADaC）がサービス窓口を開いています（**写真A-1**）．

通常，検査対象はLINに接続する装置（ECU），あるいは通信回線のドライバ回路（物理層）です．特殊な例として，ドライバ・ソフトウェア（データリンク層）を検査対象とすることも可能です．この場合，ドライバ・ソフトウェア単体では動作しないので，専用の動作プラットホーム（マイコン基板）やコンパイラ類など，開発環境一式で検査を依頼します．アプリケーションに相当するテスト・プログラム（LIN仕様のAPIに準拠してドライバ・ソフトウェアを呼び出すプログラム）はC＆Sが用意します．対象物の挙動などについて依頼者とのレビューを行いながらの作業となるため，一般にテスト期間としては4～5週間は必要です．また資料によれば，ファーストラン（初めてのテスト）で合格する確率は75％以下だそうです．



写真A-1　ADaCの作業ベンチ
現在，ADaCはECUベースと物理層単体の検査に対応している．検査結果は波形による確認があるため，人手による作業が必須とのこと．

由に使うこともできます．しかし，LINノードとして標準から外れてしまうので，仕様書上では一つ目の信号ベースの診断が推奨されています．

三つ目の方法として，LIN診断トランスポート（DIAGNOSTIC TRANSPORT LAYER）の使用が挙げられます．これは，**図11**のようなCANのバックボーン・バスにLINクラスタが接続されていて，LINスレーブでISO診断[注7]を直接実行させたい場合に用いられます．この場合，スレーブ・ノードには診断を実行できるだけの処理能力が必要となります．マスタはゲートウェイ動作を行います．

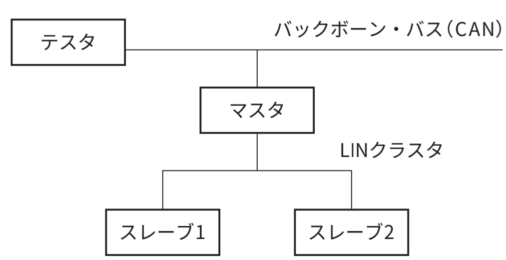


図11　診断トランスポート層使用環境の例
マスタはCAN-LINゲートウェイとして動作し，LINスレーブはCANノードと同等の診断動作を行う．

たち・のぶゆき
NECマイクロシステム（株）

<筆者プロフィール>
舘 伸幸．組み込み系ソフトウェア・エンジニア．1983年入社以来，デバイス会社でソフトウェアひとすじ．

参考・引用*文献
(1) * LIN Consortium ; LIN Specification Package Revision 2.0, Sept 2003.

注7：ISO15765-2.4，ISO15765-3.5を参照．